Intro to Python, Part 3

Functions, modules, and strings, oh my!

How This Part Will Work

I will give you a bunch of exercises

Do each one on your laptop

- Ask for help as needed
- Look up when you finish

Once everyone's done trying an exercise, I'll show the answer and explain it.

You saw the REPL

It's that interactive shell that you've been playing with.

Yes, the one that looks like this:

learn_python — Python — 70×7

```
$ python3
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Now, meet Python scripting

Python scripts are:

- Just text files
- With filenames that end in .py (not .txt)

A type of simple Python program

Hello World

The first example program for any programming language.

Traditionally used to illustrate to beginners the most basic syntax of a programming language.

Drumroll...here it is...

Exercise 0

Open a text editor

Type this:

- `print("Hello, World!")`

Save the file as ***hello_world.py***

- Where? Within a directory called ***learn_python***, which can be anywhere on your computer.

Exercise 0

Open Terminal or Powershell

Navigate to the script and run it:

learn_python — bash — 44×7

```
$ cd learn_python/
$ ls
hello_world.py
$ python3 hello_world.py
Hello, world!
$
```

Exercise 1: Your Challenge

Now write a similar program called ***birthday.py***

When you run ***birthday.py***, it should print:

- `Happy birthday, dear Carol!`

Exercise 1: The Answer

birthday.py — learn_python

birthday.py

```
1  print("Happy birthday, dear Carol!")
2
```

Line: 2 | Python | Soft Tabs: 4 | No symbols to show for current document.

learn_python — bash — 44×7

```
$ python3 birthday.py
Happy birthday, dear Carol!
$
```

Exercise 2: Your Challenge

Change ***birthday.py*** to be this program:

birthday.py — learn_python

birthday.py

```
1 def happy_birthday_carol():
2     print("Happy birthday, dear Carol!")
3
4 happy_birthday_carol()
5
```

Line: 5 | Python | Soft Tabs: 4 | happy_birthday_carol()

4 spaces
NOT tab

Then run it again to see the same output.

Study the program and try to figure out what's going on.

Output Should Be the Same

```
learn_python — bash — 44×7
$ python3 birthday.py
Happy birthday, dear Carol!
$
```

The program does the same thing, but the code was different this time.

You Just Wrote Your First Function

Indicates start of function

Name of function

birthday.py — learn_python

birthday.py

```
1 def happy_birthday_carol():
2     print("Happy birthday, dear Carol!")
3
4 happy_birthday_carol()
5
```

Line: 5 | Python | Soft Tabs: 4 | happy_birthday_carol()

Indentation matters

Calling the function executes what's inside of it

Exercise 3: Your Challenge

Modify the program to greet *both* Carol and Rise:

- Add a 2nd function to ***birthday.py*** which prints "`Happy birthday, dear Rise!`"
- Call both functions at the bottom of ***birthday.py***

Exercise 3: Recap

Whenever you run the program, you should see this:

```
learn_python — bash — 52×5
learn_python a♥d python birthday.py
Happy Birthday, dear Carol!
Happy Birthday, dear Rise!
learn_python a♥d
```

Exercise 3: The Answer

Define 2 functions and call both of them.

birthday.py — learn_python

birthday.py

```
1  def happy_birthday_carol():
2      print("Happy Birthday, dear Carol!")
3
4  def happy_birthday_rise():
5      print("Happy Birthday, dear Rise!")
6
7  happy_birthday_carol()
8  happy_birthday_rise()
9
```

Line: 9 | Python | Soft Tabs: 4 | happy_birthday_rise()

Repetition?

Notice how this program is repetitive?
We can refactor it.

birthday.py — learn_python

birthday.py

```
1  def happy_birthday_carol():
2      print("Happy Birthday, dear Carol!")
3
4  def happy_birthday_rise():
5      print("Happy Birthday, dear Rise!")
6
7  happy_birthday_carol()
8  happy_birthday_rise()
9
```

Line: 9 | Python | Soft Tabs: 4 | happy_birthday_rise()

Refactoring

Restructuring code without changing its external behavior

- You did this in Ex 2 without knowing it

We'll now refactor the 2 functions into a single function.

Exercise 4

Look carefully at this:

birthday.py — learn_python

birthday.py

```
1  def happy_birthday(name):
2      print("Happy Birthday, dear " + name + "!")
3
4  happy_birthday("Trey")
```

Line: 4:23 | Python | Soft Tabs: 4 | No symbols to show for current document.

How does it compare to what you currently have in ***birthday.py***?

What's different? What's the same?

Exercise 4

Now, go ahead and replace the text in your ***birthday.py*** file with the text here.

birthday.py — learn_python

birthday.py

```
1 def happy_birthday(name):
2     print("Happy Birthday, dear " + name + "!")
3
4 happy_birthday("Trey")
```

Line: 4:23 | Python | Soft Tabs: 4 | No symbols to show for current document.

What do you think will happen when you run it?

Exercise 4

birthday.py — learn_python

birthday.py

```
1 def happy_birthday(name):
2     print("Happy Birthday, dear " + name + "!")
3
4 happy_birthday("Trey")
```

Line: 4:23 | Python | Soft Tabs: 4

learn_python — bash — 42×5

```
$ python3 birthday.py
Happy birthday, dear Trey!
$
```

The function takes a name argument.
Then it concatenates strings.
Then it prints the final string.

Function Terminology

name is a parameter
(a variable in a function)

This is the function

birthday.py — learn_python

```
1 def happy_birthday(name):
2     print("Happy Birthday, dear " + name + "!")
3
4 happy_birthday("Trey")
```

Line: 4:23 Python Soft Tabs: 4 No symbols to show for current document.

Calling the function

"Trey" is an argument
(the value given to name)

Exercise 5: Your Challenge

Try to break `happy_birthday()` by passing in strange arguments:

- A really, really, really long name
- Numbers
- Other Python types, if you know about any other ones

This is how malicious "black hat" hackers think when they try to break into desktop and web apps.

Long Names: This Guy Is Real

This guy is a real-life test case for name fields

```
happy_birthday("Adolph Blaine Charles David Earl Frederick
Gerald Hubert Irvin John Kenneth Lloyd Martin Nero Oliver
Paul Quincy Randolph Sherman Thomas Uncas Victor William
Xerxes Yancy Zeus Wolfe-schlegelstein-hausenberger-
dorffvoraltern-waren-gewissenhaft-schaferswessen-
schafewaren-wohlgepflege-und-sorgfaltigkeit-beschutzen-
von-angreifen-durch-ihrraubgierigfeinde-welche-voraltern-
zwolftausend-jahres-vorandieerscheinen-wander-ersteer-dem-
enschderraumschiff-gebrauchlicht-als-sein-ursprung-von-
kraftgestart-sein-lange-fahrt-hinzwischen-sternartigraum-
auf-der-suchenach-diestern-welche-gehabt-bewohnbar-
planeten-kreise-drehen-sich-und-wohin-der-neurasse-von-
verstandigmen-schlichkeit-konnte-fortplanzen-und-sicher-
freuen-anlebens-langlich-freude-und-ruhe-mit-nicht-ein-
furcht-vor-angreifen-von-anderer-intelligent-geschopfs-
von-hinzwischen-sternartigraum, Senior")
```

Exercise 6

Numbers when strings are expected:

hi.py — learn_python

```
def hi(first, last):
    print("Hi " + first + last)

hi("Henry", 8)
```

Line: 4:15 Python Soft Tabs: 4 hi(first, last)

Oops! Passing in 8 instead of a string makes this fail.

Can anyone guess why?

Exercise 6: Answer

hi.py — learn_python

hi.py

```
1 def hi(first, last):
2     print("Hi " + first + last)
3
4 hi("Henry", 8)
```

Line: 4:15 | Python | Soft Tabs: 4 | hi(first, last)

learn_python — bash — 56×9

```
$ python3 hi.py
Traceback (most recent call last):
  File "hi.py", line 4, in <module>
    hi("Henry", 8)
  File "hi.py", line 2, in hi
    print("Hi " + first + last)
TypeError: Can't convert 'int' object to str implicitly
$
```

Exercise 7: String Formatting

hi.py — learn_python (git: part3-as-scripts)

```
def hi(first, last):
    print("Hi {} {}".format(first, last))

hi("Henry", 8)
```

Line: 4:15 | Python | Soft Tabs: 4 | hi(first, last)

learn_python — bash — 20×5

```
$ python3 hi.py
Hi Henry 8
$
```

str.format() **is not** the same as string concatenating!

Exercise 8: Fancy Math

factors.py — learn_python (git: part3-as-scripts)

```python
def print_factors(age):
    for i in range(1, age + 1):
        if age % i == 0:
            print(i)

print_factors(32)
```

Line: 4:17 | Python | Soft Tabs: 4 | print_factors(age)

learn_python — bash — 38×8

```
$ python3 factors.py
1
2
4
8
16
32
$
```

Exercise 9: Cupcake Tally

A good birthday party should allocate:

- 2 cupcakes per guest
- 13 cupcakes for the birthday person, because if it's your birthday, you should be allowed to eat a baker's dozen of cupcakes.

That’s how I like to plan things :)

Exercise 9: Cupcake Tally

Normally the math is too confusing, so I stick with cake

Exercise 9: Cupcake Tally

cupcakes.py — learn_python

```
def cupcake_tally(guests):
    """ Given number of party guests,
        returns how many cupcakes are needed. """
    return 2 * guests + 13

cupcakes = cupcake_tally(30)
print("You need {} cupcakes".format(cupcakes))
```

Line: 6:29 | Python | Soft Tabs: 4 | cupcake_tally(guests)

A good birthday party should allocate:

- 2 cupcakes per guest
- 13 cupcakes for the birthday person, because if it's your birthday, you should be allowed to eat a baker's dozen of cupcakes.

Exercise 9: Cupcake Tally

cupcakes.py — learn_python

```
1  def cupcake_tally(guests):
2      """ Given number of party guests,
3          returns how many cupcakes are needed. """
4      return 2 * guests + 13
5
6  cupcakes = cupcake_tally(30)
7  print("You need {} cupcakes".format(cupcakes))
```

Line: 6:29 | Python | Soft Tabs: 4 | cupcake_tally(guests)

cupcake_tally returns a value which we can print

learn_python — bash — 31×5

```
$ python3 cupcakes.py
You need 73 cupcakes
$
```

Exercise 9: Cupcake Tally

What about 2 parties:

- My 30-guest party tonight
- My private party with just Daniel tomorrow morning

How many cupcakes would we need to bake?

Exercise 9: Cupcake Tally

cupcakes.py — learn_python

```python
def cupcake_tally(guests):
    """ Given number of party guests,
        returns how many cupcakes are needed. """
    return 2 * guests + 13

cupcakes = cupcake_tally(30) + cupcake_tally(1)
print("You need {} cupcakes".format(cupcakes))
```

Line: 5 | Python | Soft Tabs: 4 | cupcake_tally(guests)

Call the function twice!

learn_python — bash — 31×5

```
$ python3 cupcakes.py
You need 88 cupcakes
$
```

Exercise 9: Cupcake Tally

cupcakes.py — learn_python

```python
def cupcake_tally(guests):
    """ Given number of party guests,
        returns how many cupcakes are needed. """
    return 2 * guests + 13

cupcakes = cupcake_tally(30) + cupcake_tally(1)
print("You need {} cupcakes".format(cupcakes))
```

Line: 5 | Python | Soft Tabs: 4 | cupcake_tally(guests)

We add the results of both cupcake_tally() calls.

learn_python — bash — 31×5

```
$ python3 cupcakes.py
You need 88 cupcakes
$
```

Summary: What are functions used for?

Program decomposition, or factoring

To break down a problem into parts.

It makes the code:

- More readable.
- Easier to understand.

Summary: What are functions used for?

Code reuse

Functions can be used instead of:

- Repeating the same lines of code at different times throughout a program.

This reduces duplication of code.

Summary: What are functions used for?

Abstraction or simplification

Using functions allows us to:

- Hide all of the details involved
- Put them into one place
- Simply call the function to execute the lines of code.

Modules

For grouping reusable code like:

- Functions
- Important constants like MAX_CUPCAKES
- Other useful stuff, e.g. decorators, context managers

Put your functions into modules for later reuse!

Exercise 10: Modules

Break our code into 2 files for reuse:

- ***cupcakes.py*** holds the function(s)
- ***main.py*** is the main program

(.py files are called 'modules')

Exercise 10: Modules

cupcakes.py

cupcakes.py — learn_python

cupcakes.py | main.py

```
1 def cupcake_tally(guests):
2     """ Given number of party guests,
3         returns how many cupcakes are needed. """
4     return 2 * guests + 13
5
```

Line: 4:13 | Python | Soft Tabs: 4 | cupcake_tally(guests)

main.py

main.py — learn_python

cupcakes.py | main.py

```
1 from cupcakes import cupcake_tally
2
3 tally = cupcake_tally(30) + cupcake_tally(1)
4 print("You need {} cupcakes".format(tally))
```

Line: 4:42 | Python | Soft Tabs: 4 | Symbols

Exercise 10: Modules

```
learn_python — bash — 48×5
$ python3 main.py
You need 88 cupcakes
$
```

- ***main.py*** imports `cupcake_tally` from ***cupcakes.py***
- ***main.py*** calls the `cupcake_tally` function twice
- ***main.py*** sums the returned value of both function calls and prints the result.

Wait, there's more!

You can also find and use modules from:

- Python standard library (stdlib)
- Python Package Index (PyPI)

I'll cover those at the end of the day, in the conclusion.